able), and the related algorithms are often tricky.

In the WAM research team we have developed an XML/XPath static analyser based on a new logic of finite trees. This analyser consists in compilers that allow XML types and XPath queries to be translated into this logic, and an optimized logical solver for testing satisfiability of a formula of this logic.

The benefit of these compilers is that they allow one to reduce all the problems listed above, and many others, to logical satisfiability. This approach has a couple of important practical advantages. First of all, one can use the satisfiability algorithm to solve all of these problems. More importantly, one could easily explore new variants of these problems, generated for example by the presence of different kinds of type or schema information, with no need to devise a new algorithm for each variant.

The system has been implemented in Java and uses symbolic techniques (binary decision diagrams) in order to enhance its performance. It is capable of comparing path expressions in the presence of real-world DTDs (such as the W3C SMIL and XHTML language recommendations). The cost ranges from several milliseconds for comparison of XPath queries without tree types, to several seconds for queries under very large, heavily recursive, type constraints, such as the XHTML DTD. These preliminary measurements shed light for the first time on the cost of solving static analysis problems in practice. Furthermore, the analyser generates XML counter-examples that allow program defects to be reproduced independently from the analyser.

The development of these analysers was initiated by the Web Adaptation and Multimedia research team at INRIA, Grenoble - Rhône-Alpes Research Centre, France. The project commenced in October 2005 and the full system will soon be released publicly.

Links:
Project home page:
http://wam.inrialpes.fr/xml
WAM team: http://wam.inrialpes.fr

Please contact:
Pierre Genevès
WAM Team, CNRS, France
Tel: +33 4 76 61 53 84
E-mail: Pierre.Geneves@inria.fr

Nabil Layaïda
WAM Team, INRIA, France
Tel: +33 4 76 61 52 81
E-mail: Nabil.Layaida@inria.fr

Seaside – Advanced Composition and Control Flow for Dynamic Web Applications

by Alexandre Bergel, Stéphane Ducasse and Lukas Renggli

Page-centric Web application frameworks fail to offer adequate solutions to model composition and control flow. Seaside allows Web applications to be developed in the same way as desktop applications. Control flow is modelled as a continuous piece of code, and components may be composed, configured and nested as one would expect from traditional user interface frameworks.

Seaside is a framework for building dynamic Web applications, uniquely combining object-oriented and continuation-based approaches. Seaside applications are built by composing stateful components, each encapsulating a small portion of the page. Programmers are freed from the concern of providing unique names, since Seaside automates this by associating callback functions with links and form fields. Control flow is expressed as a continuous piece of code and in addition, Seaside offers a rich application programming interface (API) to integrate with the latest Web 2.0 technology, such as AJAX (Asynchronous JavaScript) and Comet (Server Push).

Seaside is implemented in Smalltalk, a dynamically typed programming language. Seaside inherits powerful reflective capabilities from the underlying language. Web applications may be debugged while the application is running. Inspection and modification may occur on objects on the fly. Source code is changeable and recompilable without interrupting the running application, and there is no need to restart a session.

Whereas most other Web application frameworks work in a page-centric fashion, Seaside makes use of stateful components that encapsulate a small portion of a page. Developers can compose the user interface as a tree of individual components, and often these components are reused over and over again, within and between applications. A basic set of ready-made widgets is also provided to handle user interactions.

Seaside offers a mechanism by which objects may be registered to be backtracked. With every response sent to the client, Seaside takes a snapshot of and caches registered objects. This allows previous application states to be restored in a controlled fashion, for example when the user is using the 'back' button in the Web browser.

Seaside uses programmatic XHTML generation. Instead of repeatedly pasting the same sequence of tags into templates, it provides a rich API to generate XHTML. This approach not only avoids common problems with invalid markup, but also allows markup patterns to be easily abstracted into convenient reusable methods. CSS (Cascading Style Sheets) are used to give the application a professional look.

Seaside also provides callback-based request handling. This allows developers to associate a piece of code with anchors and form fields, which are then automatically performed when the link is clicked or the form is submitted. This feature makes it almost trivial to connect the view with its model, as Seaside abstracts all serialization and parsing of query parameters away.

Callbacks are used in a natural way to define a control flow, for example to temporarily delegate control to a sequence of other components. The flow is defined by writing plain source code. Control statements, loops and method calls are mixed with messages to display components. Whenever a new view is generated, the control flow is suspended and the response is sent back to the client. Upon a new user interaction the flow is resumed.

In order that the definition of control flows as part of a Web application be as seamless as possible, Seaside internally stores a 'continuation' whenever a new component is displayed. This suspends the current control flow and allows one to resume it later on. Since continuations may be resumed multiple times, Seaside supports the use of Web browsers' 'back' and 'forward' buttons at any time. The execution state is automatically restored to the requested point and everything behaves as the developer expects.

To complement the expressiveness of the Smalltalk programming language, a set of tools including a memory analyser, a speed profiler, a code editor and an object inspector are included. The debugger supports incremental code recompilation, and enables highly interactive Web applications to be built quickly and in a reusable and maintainable fashion.

Seaside is open-source software distributed under the MIT licence. It is under constant development by an international community using Squeak Smalltalk. Due to the platform independence of Smalltalk, Seaside can be run on almost any platform. The Seaside application server can be bridged with industrial scale servers, such as Apache. Seaside is supported by major commercial Smalltalk vendors, GemStone Smalltalk and Cincom VisualWorks, as part of their product strategy, and is widely used in an industrial context.

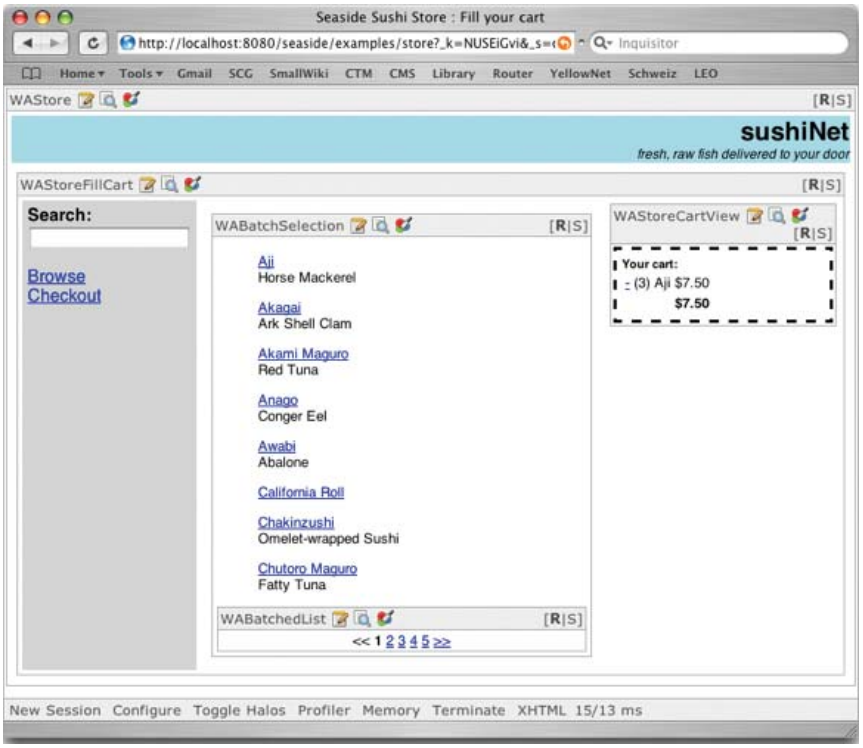

Figure 1: A Web application built with Seaside.

The official Seaside Web site uses Pier, an open-source content management system written on top of Seaside.

Link:
http://www.seaside.st

Please contact:
Alexandre Bergel
INRIA Futurs, Lille, France
E-mail: alexandre.bergel@inria.fr

EAGER: A Novel Development Toolkit for Universally Accessible Web-Based User Interfaces

by Constantina Doulgeraki, Alexandros Mourouzis and Constantine Stephanidis

EAGER is an advanced toolkit that helps Web developers to embed in their artefacts accessibility and usability for all. Web applications developed by means of EAGER have the ability to adapt to the interaction modalities, metaphors and user interface elements most appropriate to each individual user and context of use.

The constantly evolving Web is an unprecedented and continuously growing source of knowledge, information and services, potentially accessible by anyone, at any time and from anywhere. Despite the universality of the Web and the predominant role of Web-based user interfaces in the evolving Information Society, current approaches to Web design do not embrace the notion of adaptation and the principles of 'design for all'. Consequently, they fail to satisfy the individual interaction needs of target users with different characteristics. A common practice in contemporary Web development is to deliver a single user-interface design that meets the requirements of an 'average' user. However, this 'average' user is in fact an imaginary entity, and differs radically from the profiles of a large portion of the population. This is particularly the case for people with a disability, elderly people, novice IT users and users on the move.